GALAMOST

GPU-Accelerated Large-Scale Molecular Simulation Toolkit

USER MANUAL

Version 0.0.3

Contents

1 Introduction 1
1.1 A sketch of GALAMOST 1
2 Installation and Usage 3
2.1 Compiling and installation 3
2.2 Usage by script 3
2.3 Usage by user graphic interface 6
3 Force Fields 7
3.1 Short range non-bonded forces 7
3.2 Bonded forces 8
3.3 Numerical forces 10
4 Characteristic Models 12
4.1 Soft anisotropic particle model 12
4.2 Polymerization model 13
4.3 Gay-Berne particle model 13
5 Configuration Input 15
5.1 Coordinate file 15
5.2 Topologies 16
6 Command Index 18
6.1 Reading the initial configuration of system 18
6.1.1 Xml parser 18
6.1.2 Bin parser 18
6.2 Non-bonded forces 18
6.2.1 LJ force 18
6.2.2 DPD force 19
6.2.3 Pair force for LJ12_6, LJ9_6, harmonic, Gauss 20
6.2.4 External force 21
6.3 Bonded forces 22
6.3.1 Bond harmonic force 22
6.3.2 Bond fene force 23

6.3.3 Bond polynomial force 23
6.3.4 Angle harmonic force 24
6.3.5 Angle cosine harmonic force 24
6.3.6 Angle cosine force 25
6.3.7 Dihedral harmonic force 26
6.3.8 Dihedral Opls Cosine force 26
6.4 Integration method 27
6.4.1 Nve ensemble 27
6.4.2 Nvt ensemble with Nose Hoover method 27
6.4.3 Nvt ensemble with Berendsen method 28
6.4.4 Nvt ensemble with Andersen method 28
6.4.5 Npt ensemble with Andersen barostat 29
6.4.6 Nve rigid body thermostat 30
6.4.7 Nvt rigid body thermostat 30
6.4.8 Brownian dynamic nvt rigid body thermostat 31
6.4.9 Nve rigid body thermostat with tunable freedoms 31
6.5 Particle Group 32
6.5.1 Build group by keywords 32
6.5.2 Build group by index range of particles 33
6.5.3 Combine groups 33
6.6 Particle List 33
6.6.1 Build neighbor list 33
6.6.2 Build cell list 34
6.7 Data dump 35
6.7.1 The dump of some important informations 35
6.7.2 Configuration dump with mol2 format 35
6.7.3 Configuration dump with xml format 36
6.7.4 Configuration dump with dcd format 38
6.7.5 Configuration dump with binary format 38
6.8 Some other functions 39
6.8.1 Removing the momentum of center of mass 39

6.8.2 The computition of some important informations 39
6.8.3 Sorting data 40
6.8.4 VariantLinear 40
6.8.5 VariantSin 41
6.8.6 VariantWell 41
6.9 Some functional modules 42
6.9.1 The coarse-graining numerical method 42
6.9.2 The MD-SCF method 44
6.9.3 The Polymerization method 45
6.9.4 Anisotropic model 47
6.9.5 Gay-Berne model 48
6.10 Configuration generator 50
6.11 Data tackle 55
7 Appended Documents 58
7.1 Polymerization model 58

1 Introduction

1.1 A sketch of GALAMOST

GALAMOST is a versatile molecular simulation package which is designed to utilize computational power of graphics processing units (GPUs) as much as possible. In addition to common features of molecular dynamics programs, it is developed specially for coarse-graining simulation of polymeric systems by encompassing some latest techniques, such as hybrid particle-field molecular dynamics, iterative Boltzmann inversion numerical potential method, soft anisotropic particle model, and chain-growth polymerization model. By continuously optimizing the algorithms, each method implemented on a single GPU by GALAMOST has gained a good performance. Some biomolecules and polymers can be modeled in coarse-grained MD (CGMD) simulations implemented by GALAMOST. The CGMD has become a powerful tool, accounting for the problems related to self-assembly, phase separation, and other phenomena of polymeric systems.

The choices for general force field functions in GALAMOST will be listed in the following sections. In addition, a hybrid particle field MD technique (MD-SCF) for calculating intermolecular interactions has been incorporated in GALAMOST. With this technique, the most computational time-consuming parts in MD, i.e., the intermolecular pair interactions, are replaced by interactions of particles with density fields. It will greatly speed up some slowly evolving collective processes in MD simulations, such as micro-phase separation and self-assembly of polymeric systems. In addition to analytical potentials, numerical potential can be used in GALAMOST by reading the potential table derived from iterative Boltzmann inversion (IBI) method or other structure-based coarse-graining methods. The IBI method derives the coarse-grained potentials by mapping the structural distributions onto the ones obtained either from atomistic simulations or from experiments. With this bottom-up coarse-graining scheme, the derived coarse-grained numerical

potentials can be applied in larger systems, but under the same thermodynamic conditions.

Besides some basic functions of general MD, such as CGMD, Brownian dynamics (BD), and dissipative particle dynamics (DPD), GALAMOST also encompasses several tailor-made modules: a soft anisotropic particle model has been incorporated for modeling some kinds of anisotropic particles and a stochastic chain-growth polymerization reaction model has been developed specially for the studies related to polymerization. By using these advanced simulation techniques on the GPU, GALAMOST enriches the routes for researchers to investigate polymeric systems via computer simulations.

2 Installation and Usage

2.1 Compiling and installation

The entire GALAMOST package is a Free Software under the GNU General Public License. The package is mainly distributed as source code and binary program. The binary program of full version and the code of simplified version can be downloaded from our website *www.galamost.com*. The full version of code can be gotten by emailing to corresponding authors.

Here is the guide for installation by code. Before compiling and installing of source code, you should set configuration firstly, i.e. installation path, by following command.

./configuration --prefix=/opt/galamost

More configuration options are given here,

Commands	Functions	Examples
--prefix=	installation path	*--prefix=/opt/galamost*
--cuda_arch=	Compute capability of GPU	*--cuda_arch=20*
--precision=	precision format	*--precision=double*
--gprof=	profiling tool	*--gprof=on*
--gdb=	gdb tool	*--gdb=on*

After configuring, a Makefile will be generated in current directory. Then you can compile and install the package by the following command,

make install -j4

where *-j* indicates the number of threads to compile the code.

2.2 Usage by script

The C++ and CUDA C code has been written as an extended module of Python. We can use the functions of GALAMOST by loading and calling its modules through a Python script.

With the prepared script, you should now be able to try running GALAMOST as:

./dpd.gala --gpu=0 >a.log&

where you may specify the GPU id with the "--gpu=" option and output the screen information into "a.log" file. If the script file has no executive permission, the command of "chmod +x dpd.gala" should be executed before running above command.

Here is an example of script. The head of GALAMOST script such as "dpd.gala" usually is:

```
#!/usr/bin/python
import sys
sys.path.append('/opt/galamost/lib')
import galamost

global _options
parser = OptionParser()
parser.add_option('--gpu', dest='gpu',help='GPU on which to execute')
(_options, args) = parser.parse_args()
```

where the first paragraph sets the path of the installed library of GALAMOST (galamost.so) for loading the Python extensible modules of GALAMOST. The second paragraph is used for parsing GPU id from the executive command.

Then, by reading the configuration from a prepared XML file, a system with necessary information can be built up by:

```
filename = 'A.xml'
build_method = galamost.XmlReader(filename)
perform_config = galamost.PerformConfig( int(_options.gpu))
all_info = galamost.AllInfo(build_method,perform_config)
```

After that, we need to build up an application which will call following defined and added objects by:

```
dt = 0.01
app = galamost.Application(all_info, dt)
```

Further, we should define the needed objects by the classes of GALAMOST and pass them to the application, such as the following example: non-bonded DPD force, NVT thermostat with GWVV algorithm, and information analysis methods:

```
neighbor_list = galamost.NeighborList(all_info, 1.0 ,0.05)#(,rcut,rbuffer)
dpd=galamost.DpdForce(all_info, neighbor_list, 1.0, 12345)#(,,rcut,seed)
dpd.setParams('A', 'A', 25.0, 3.0)#(,,alpha,sigma)
dpd.setParams('A', 'B', 40.0, 3.0)#( ,,alpha,sigma)
dpd.setParams('B', 'B', 25.0, 3.0)#( ,,alpha,sigma)
app.add(dpd)

group = galamost.ParticleSet(all_info, "all" )
comp_info = galamost.ComputeInfo(all_info, group)
Gwvv = galamost.DpdGwvv(all_info, group)
app.add(Gwvv)

dinfo = galamost.DumpInfo(all_info, comp_info, 'data.log')
dinfo.setPeriod(200)
app.add(dinfo)
```

The tail of script usually sets the freedom degree for "ComputeInfo" method, the number of time steps to run, and the function of analysis of neighbor list etc.

```
comp_info.setNdof(app.getNdof())
app.run( 10000)
neighbor_list.printStats()
```

2.3 Usage by user graphic interface

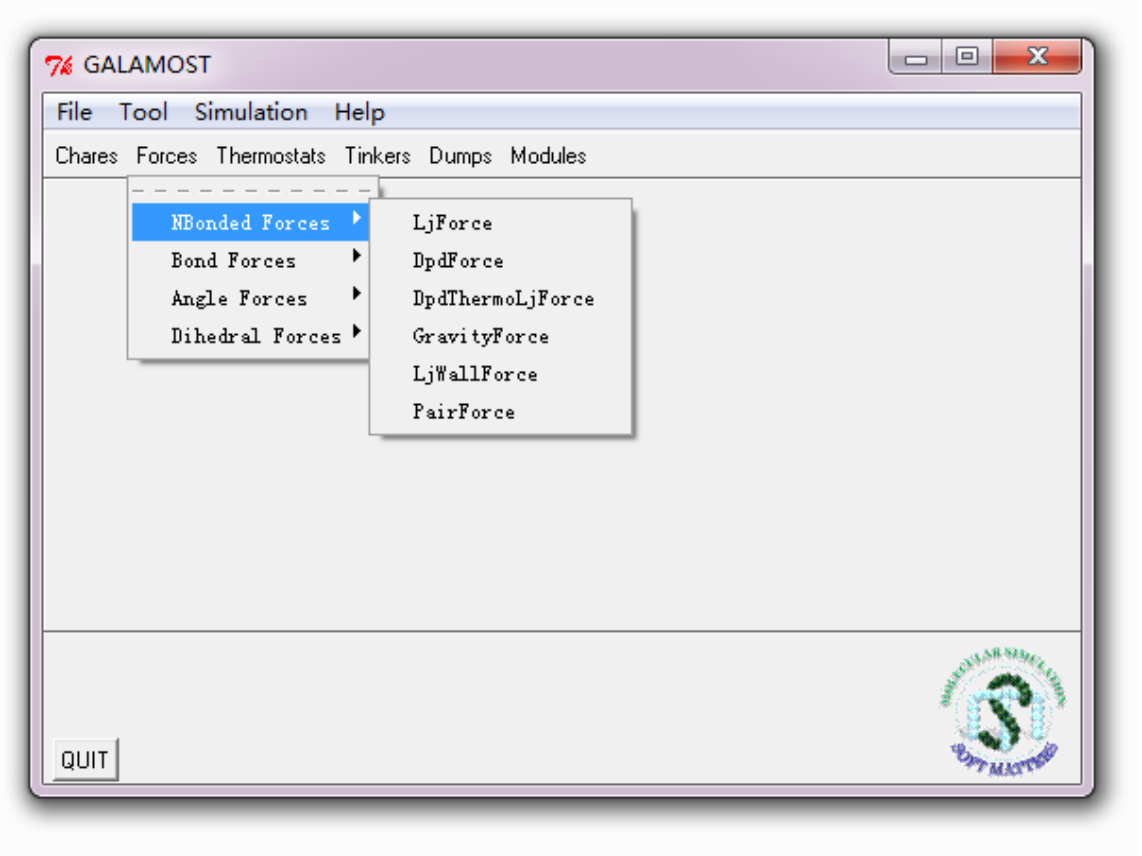

Figure 2.1 The snapshot of UGI

User graphic interface (UGI) can directly run and display on personal computer with Windows operation system and installed Python. UGI also can run on remote workstation with Linux operation system and display on terminal computer with Windows through Exceed or some other X server programs.

UGI is usually used to generate GALAMOST script. Before doing that, we should choose a XML file which contains the necessary information of the system, (i.e. particle types and bond types) to help building script. We can select the functions of GALAMOST and set parameters by clicks and inputs. By building a project, a window will be open to indicate the chosen items. With it, you can cancel or reset parameters. At last, you can generate and output a GALAMOST script.

3 Force Fields

3.1 Short range non-bonded forces

In polymer systems, the net non-bonded force of each particle is produced by summing all the non-bonded forces of neighboring particles. The standard algorithm is to use a neighbor list that lists the interacting particles for each particle, built beforehand. Because of the independence of parallel CUDA threads, a pair of interacting particles is inevitably included independently in neighbor list in the mode that one thread calculates and sums all non-bonded forces of a particle. The common non-bonded potential energy functions are included in GALAMOST, which are

Lennard-Jones (LJ) (CI 6.2.1):

$$\phi(r) = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \alpha\left(\frac{\sigma}{r}\right)^{6}\right) \tag{3.1}$$

where ϵ is the depth of the potential well, σ is the finite distance at which the interparticle potential is zero, and r_{ij} is the distance between the particles;
Harmonic repulsion (CI 6.2.3):

$$U_{harmonic}(r_{ij}) = \frac{1}{2}\alpha\left(1 - \frac{r_{ij}}{r_{cut}}\right)^{2} \tag{3.2}$$

where α and rcut set the maximum energy penalty and maximum interaction distance, respectively;

Gaussian repulsion (CI 6.2.3):

$$U_{Gaussion}(r_{ij}) = \varepsilon \exp\left[-\frac{1}{2}\left(\frac{r_{ij}}{\sigma}\right)^{2}\right] \tag{3.3}$$

where ε and σ determine the energy and length scales, respectively.

3.2 Bonded forces

The bonded forces generally include the bond, angle, and torsion forces. Because of the execution mode of SIMD in GPU, it is efficient that a thread calculates and sums the bond, angle, and torsion forces of a particle in the kernel functions. Therefore, we can employ the thread whose number is equal to the number of particle to calculate the bonded forces of each particle independently. Although the force of a bond will be computed twice (three times for an angle and four times for a torsion) on device, it casts the computation well into parallel mode for separated threads of GPUs and is efficient for data copy between host and device memories. The common bonded potential energy functions in GALAMOST are

Bond harmonic (CI 6.3.1):

$$U_{bond}(r_{ij}) = \frac{1}{2} k^{bond} (r_{ij} - r_0)^2 \tag{3.4}$$

Bond FENE (CI 6.3.2):

$$U_{bond}(r_{ij}) = -\frac{1}{2} k^{bond} r_m{}^2 \log(1 - \frac{r_{ij}{}^2}{r_m{}^2}) \tag{3.5}$$

Bond polynomial (CI 6.3.3):

$$U_{bond}(r_{ij}) = k_1{}^{bond} (r_{ij} - r_0)^2 + k_2{}^{bond} (r_{ij} - r_0)^4 \tag{3.6}$$

in which r_{ij} is the instantaneous length of the bond, r_0 is the equilibrium length of the bond, r_m is maximum length, and k^{bond} is the spring constant;

Angle harmonic (CI 6.3.4):

$$U_{angle}(\theta_{ijk}) = \frac{1}{2} k^{angle} (\theta_{ijk} - \theta_0)^2 \tag{3.7}$$

Angle cosine harmonic (CI 6.3.5):

$$U_{angle}(\theta_{ijk}) = \frac{1}{2} k^{angle} \left\{ \cos\left(\theta_{ijk}\right) - \cos\left(\theta_0\right) \right\}^2 \tag{3.8}$$

Angle cosine (CI 6.3.6):

$$U_{angle}(\theta_{ijk}) = k^{angle} \left[1 - \cos\left(\theta_{ijk} - \theta_0\right) \right] \tag{3.9}$$

in which θ_{ijk} is the angle in radians between vectors r_{ij} and r_{jk}, θ_0 is the equilibrium angle, and k^{angle} is the angle force constant;

Dihedral cosine (CI 6.3.7):

$$U_{dihedral}(\varphi_{ijkl}) = k^{dihedral} \left[1 - \cos\left(\varphi_{ijkl} - \delta\right) \right] \tag{3.10}$$

Dihedral Opls (CI 6.3.8):

$$U_{dihedral}(\varphi_{ijkl}) = k_1^{dihedral} + k_2^{dihedral} \left[1 + \cos\left(\varphi_{ijkl} - \delta\right) \right] + k_3^{dihedral} \left[1 - \cos\left(2\varphi_{ijkl} - 2\delta\right) \right] + k_4^{dihedral} \left[1 + \cos\left(3\varphi_{ijkl} - 3\delta\right) \right] \tag{3.11}$$

in which ϕ_{ijkl} is the angle in radians between the planes *(i,j,k)* and *(j,k,l)* which is drawed in Fig. 3.1, δ is the phase shift angle, and $k^{dihedral}$ is the multiplicative constant.

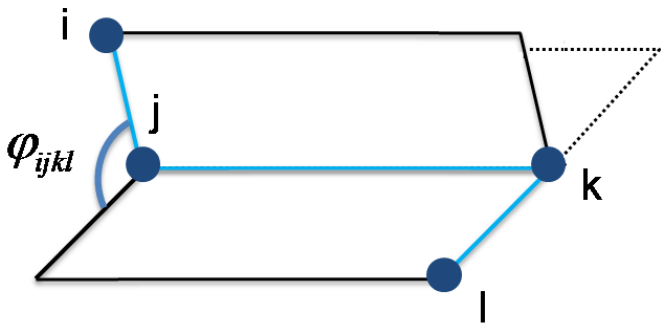

Figure 3.1 The schematic diagram of ϕ_{ijkl}

3.3 Numerical forces

The numerical non-bonded, bond, angle, and torsion potentials can be derived from IBI or reverse Monte Carlo method. With IBI method, the procedure starts with the potentials of mean force as guessed potentials and then optimizes the potentials iteratively by mapping the structural distributions (i.e., radial distribution function, RDF) onto the ones obtained either from atomistic simulations or from experiments. The resulting numerical potentials usually take the form as a table in which the potential values at discrete grid points of distance are given. In the treatment of tabulated potentials, the initial inputted potential tables on grid points of r are transformed to the tables (arrays) on grid points of $z = r^2$. With this trick, the $r = SQRT(r^2)$ in the inner loop of force calculation is avoided, and the force is then calculated by

$$F = -\mathbf{r}\frac{\partial V(r)}{\partial r}\frac{1}{r} = -2\mathbf{r}\frac{\partial V(z)}{\partial z} \tag{3.12}$$

Within each interval between the grid points, potentials are fitted to a cubic spline function, more specifically, for each $x_i < x < x_{i+1}$, let $\delta = x - x_i$, $V(x)$ is represented by

$$V(x) = c_0 + c_1\delta + c_2\delta^2 + c_3\delta^3 \tag{3.13}$$

where x corresponds to z, θ, and φ for particle-particle distance square, bending angle, and torsion angle, respectively. i is the index of the grid point and c_0 is the starting potential value of each grid point. Other parameters c_1 , c_2 , and c_3 are chosen to make the values of the first derivative and the second derivative at both ends of interval x_i and x_{i+1} equal to the correct values of function V.

4 Characteristic Models

4.1 Soft anisotropic particle model

By adding two degrees of freedom of rotation, our one-site anisotropic particle model can be used to describe disk-like, rod-like, diblock and triblock Janus particles. We have successfully examined the packing and the self-assembly of anisotropic particles with this model.

In the simulations of disk-like and rod-like particles, we adopt a soft anisotropic potential on the basis of the conservative potential in DPD. It can be expressed as,

$$U_{ij} = (1-\mu f^{\nu})\frac{\alpha_{ij}}{2}(1-r_{ij})^2 \tag{3.14}$$

where the magnitude of α_{ij} controls the strength of repulsion, μ controls the shape of the particles, and υ controls the angular width of repulsion. The disk-like or rod-like particle can be described by different expressions of anisotropic factor *f*. In disk-like particle model, the anisotropic factor is

$$f = \frac{(\mathbf{n}_i \cdot \mathbf{r}_{ij})(\mathbf{n}_j \cdot \mathbf{r}_{ij})}{r_{ij}^{\,2}} \tag{3.15}$$

where $\mathbf{n}_i$ and $\mathbf{n}_j$ are unit vectors assigning the orientations to particles *i* and *j*, respectively. $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$ is the interparticle vector.

In rod-like particle model, the anisotropic factor is

$$f = \sin\theta_i \sin\theta_j \tag{3.16}$$

where θ_i is the angle between $\mathbf{n}_i$ and the interparticle vector $\boldsymbol{r}_{ji} = -\boldsymbol{r}_{ij}$, θ_j is the angle between $\mathbf{n}_j$ and r_{ij}.

4.2 Polymerization model

In this model, we consider free radical linear chain-growth polymerization, that is, $\mathrm{mA} \rightarrow (-\mathrm{A}-)_{\mathrm{m}}$ for monomer A. Polymerization probability p_r is set to determine whether a monomer will react with an active end or not in a reaction step and is coupled to the real reaction rate r_p by

$$r_p = -\frac{d[M]}{dt} = \frac{[P^*]P_r}{\tau} \tag{3.17}$$

where $[M]$ is the free monomer concentration, $[P^*]$ is the concentration of growth centers, and τ is the reaction time interval. In a time interval, if a polymerization reaction event takes place between a monomer and an active end, a bond connection should be added between them and then the active end should be transferred to the new end.

4.3 Gay-Berne particle model

This model is usually used to describe ellipsoidal particles. Except for translocation, the rotation of these anisotropic particles has to be considered. Then, forces and torques are typically incorporated into molecular dynamics simulation. The forces and torques are derived from Gay-Berne anisotropic potential which is given here

$$V = 4\epsilon\left[\frac{\sigma_0^{12}}{\left(r-\sigma+\sigma_f\right)^{12}} - P_s\frac{\sigma_0^{6}}{\left(r-\sigma+\sigma_f\right)^{6}}\right] \tag{3.18}$$

$$\epsilon = \epsilon_0\epsilon_s\epsilon_1^{\nu}\epsilon_2^{\mu} \tag{3.19}$$

$$\epsilon_1 = \left[1 - \chi^2 \left(\hat{u}_1 \cdot \hat{u}_2\right)^2\right]^{-\frac{1}{2}} \tag{3.20}$$

$$\epsilon_2 = 1 - \frac{\chi'}{2}\left[\frac{\left(\hat{r} \cdot \hat{u}_1 + \hat{r} \cdot \hat{u}_2\right)^2}{1 + \chi'\left(\hat{u}_1 \cdot \hat{u}_2\right)} + \frac{\left(\hat{r} \cdot \hat{u}_1 - \hat{r} \cdot \hat{u}_2\right)^2}{1 - \chi'\left(\hat{u}_1 \cdot \hat{u}_2\right)}\right] \tag{3.21}$$

$$\sigma = \sigma_s \left[1 - \frac{\chi}{2}\left(\frac{\left(\hat{r} \cdot \hat{u}_1 + \hat{r} \cdot \hat{u}_2\right)^2}{1 + \chi\left(\hat{u}_1 \cdot \hat{u}_2\right)} + \frac{\left(\hat{r} \cdot \hat{u}_1 - \hat{r} \cdot \hat{u}_2\right)^2}{1 - \chi\left(\hat{u}_1 \cdot \hat{u}_2\right)}\right)\right]^{-\frac{1}{2}} \tag{3.22}$$

$$\sigma_f = \sigma_0 \left(P_s\right)^{1/6} \tag{3.23}$$

$$\chi = \frac{\left(\sigma_e / \sigma_s\right)^2 - 1}{\left(\sigma_e / \sigma_s\right)^2 + 1} \tag{3.24}$$

$$\chi' = \frac{1 - \left(\epsilon_e / \epsilon_s\right)^{1/\mu}}{1 + \left(\epsilon_e / \epsilon_s\right)^{1/\mu}} \tag{3.25}$$

where σ_e is end-to-end length, σ_s is side-to-side length, ϵ_e is end-to-end energy, and ϵ_s is side-to-side energy.

5 Configuration Input

5.1 Coordinate file

We take XML format file as the standard inputting configuration file. The **.xml* file can contain types, masses, coordinates, velocities, topological information, and so on.

Here is an example of XML file of 4 particles system. The position data in a line (x, y, and z directions) locates a particle in box and one line corresponds to one particle in sequence. This is also same rule for some other data, but do not include topological information.

```
<?xml version="0.2" encoding="UTF-8"?>
<galamost_xml>
<configuration time_step="0">
<box units="sigma"    Lx="10" Ly="10" Lz="10"/>
<position units="sigma">
-1 2 -1
-2 3 0
-1 4 1
-1 5 2
</position>
<velocity units="sigma/tau">
1 2 3
1 0 0
3 -2 1
```

0 1 1

</velocity>

<type>

A

B

B

A

</type>

<mass>

1.0

2.1

1.0

1.0

</mass>

</configuration>

</galamost_xml>

5.2 Topologies

<bond>

polymer 0 1

polymer 1 2

polymer 2 3

</bond>

Topological information also can be inputted to program by XML file. As

above example, each bond connection should be given in one line. In addition to bond connection, angle and dihedral information also should be given explicitly in XML as following format, if they are needed.

<angle>

theta 0 1 2

theta 1 2 3

</angle>

<dihedral>

phi 0 1 2 3

</dihedral>

6 Command Index

6.1 Reading the initial configuration of system

6.1.1 Xml parser

Constructor:

XmlReader(const std::string &)

#note: specifies the XML file parser method

#parameter: XML file name

Example:

filename = 'dppc.xml'

#sets the name of inputting XML file

build_method = galamost.XmlReader(filename)

#builds up the reading object of inputting XML file

6.1.2 Bin parser

Constructor:

BinaryReader(const std::string &)

#note: specifies the GALAMOST binary file parser method

#parameter: binary file name

Example:

filename = 'initial.bin'

#sets the name of GALAMOST binary file

build_method = galamost.BinaryReader(filename)

#builds up the reading object of inputting GALAMOST binary file

6.2 Non-bonded forces

6.2.1 LJ force

Constructor:

LjForce(boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>, Real)

#note: specifies the LJ pair force

#parameters: AllInfo, NeighborList, cut-off radius of interaction

Functions:

void setParams(const std::string &, const std::string &, Real, Real, Real)

#note: specifies the LJ interaction parameters by pair types

#parameters: type 1, type 2, epsilon, sigma, alpha

void setParams(const std::string &, const std::string &, Real, Real, Real, Real)

#note: specifies the LJ interaction parameters by pair types

#parameters: type 1, type 2, epsilon, sigma, alpha, cut-off of radius

void setEnergy_shift()

note: calls the function to shift LJ potential to be zero at the cut-off point

Example:

```
lj = galamost.LjForce(all_info, neighbor_list, 3.0)
#initializes LjForce object by: AllInfo object, NeighborList object, cut-off
lj.setParams('A', 'A' , 1.0, 1.0, 1.0)
#sets parameters: particle type 1, particle type 2, epsilon, sigma, alpha
lj.setEnergy_shift()
#shifts LJ potential to be zero at cut-off
app.add(lj)
#adds this object to the application
```

6.2.2 DPD force

Constructor:

DpdForce(boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>,
Real, Real, unsigned int)

#note: specifies the DPD pair force

#parameters: AllInfo, NeighborList, cut-off of radius, temperature, RNG seed

DpdForce(boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>,
Real, unsigned int)

#note: specifies the DPD pair force (the defalt temperature is 1.0)

#parameters: AllInfo, NeighborList, cut-off of radius, RNG seed

Functions:

void setParams(const std::string &, const std::string &, Real, Real)

#note: specifies the DPD interaction parameters by pair types

#parameters: type1, type2, alpha, sigma

void setT(Real)

#note: specifies the temperature

#parameters: a constant value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a varing temperature according to time steps

void setDPDVV()

#note: calls the function to enable DPDVV method (the default is GWVV)

Example:

dpd = galamost.DpdForce(all_info, neighbor_list, 1.0, 12345)

#initializes DpdForce object by: AllInfo object, NeighborList object, cut-off, RNG seed

dpd.setParams('A', 'A', 25.0, 3.0)

#sets parameters by: particle type 1, particle type 2, alpha, sigma

app.add(dpd)

#adds this object to the application

6.2.3 Pair force for LJ12_6, LJ9_6, harmonic, Gauss

Constructor:

PairForce (boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>)

#note: specifies the pair force

#parameters: AllInfo, NeighborList

Functions:

void setParams(const std::string &, const std::string &, Real, Real, Real, Real, Func)

#note: specifies the pair interaction type and parameters by pair types

#parameters: type1, type2, parameter0, parameter1, parameter2, cut-off, function type

Function type	Parameter 0	Parameter 1	Parameter 2

lj12_6	epsilon	sigma	alpha
lj9_6	epsilon	sigma	alpha
harmonic	alpha		
gauss	epsilon	sigma	

Example:

```
pair = galamost. PairForce (all_info, neighbor_list)
#initializes PairForce object by: AllInfo object, NeighborList object
pair.setParams('A', 'A', 1.0, 1.0, 1.0, 3.0, galamost.PairForce.Func. lj12_6)
#sets parameters by: type 1, type 2, epsilon, sigma, alpha, LJ12_6 function
pair.setParams('A', 'B', 1.0, 1.0, 1.0, 3.0, galamost.PairForce.Func. lj9_6)
#sets parameters by: type 1, type 2, epsilon, sigma, alpha, LJ9_6 function
pair.setParams('B', 'B', 1.0, 1.0, 1.0, 3.0, galamost.PairForce.Func. lj12_6)
#sets parameters by: type 1, type 2, epsilon, sigma, alpha, LJ12_6 function
app.add(pair)
#adds this object to the application
```

6.2.4 External force

Constructor:

```
ExternalForce(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>);
#note: specifies the external force for the group particles
#parameters: AllInfo, ParticleSet
```

Functions:

```
void setForce (boost::shared_ptr<Variant>, std::string)
#note: specifies the force magnitude varing by time steps and direction
#parameters: Variant and direction (candidates are “X”, “Y”, and “Z”)
void setForce (boost::shared_ptr<Variant>, Real, Real, Real)
#note: specifies the force magnitude varing by time steps and direction
#parameters: Variant and directional vector with x, y, and z
void setParams(const std::string &, Real)
#note: specifies the factor multiplied by the external force
#parameters: particle type, factor
```

void setParams(unsigned int, Real)

#note: specifies the factor multiplied by the external force

#parameters: particle index, factor

Example:

```
v = galamost. VariantSin ()
v.setPoint(0, 1000, 1, -1)
v.setPoint(1000000, 1000, 1, -1)
#set the parameters of sinesoid force by timestep, period, max and min value.
The latter three paramters are linearly varing by timestep
groupA = galamost.ParticleSet(all_info,”A”)
ef = galamost. ExternalForce (all_info, groupA)
#initializes ExternalForce object by: AllInfo object, ParticleSet object
ef. setForce (v, “X”)
#sets parameters by: force and direction
app.add(ef)
#adds this object to the application
```

6.3 Bonded forces

6.3.1 Bond harmonic force

Constructor:

BondForceHarmonic(boost::shared_ptr<AllInfo>)

#note: specifies the harmonic bond force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real)

#note: specifies the harmonic bond interaction parameters of different types

#parameters: bond type, spring constant, equilibrium length

Example:

```
bondforce = galamost.BondForceHarmonic(all_info)
#initializes BondForceHarmonic object by: AllInfo object
bondforce.setParams('polymer', 1250.000, 0.470)
```

#sets parameters by: bond type, spring constant, equilibrium bond length

app.add(bondforce)

#adds this object to the application

6.3.2 Bond fene force

Constructor:

BondForceFene (boost::shared_ptr<AllInfo>)

#note: specifies the FENE bond force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real)

#note: specifies the FENE bond force parameters

#parameters: bond type, spring constant, maximum length of the bond

void setParams(const std::string &, Real, Real , Real, Real)

#note: specifies the FENE+LJ bond parameters

#parameters: bond type, spring constant, maximum length of the bond,

epsilon, sigma

Example:

bondforcefene = galamost.BondForceFene(all_info)

#initializes BondForceFene object by: AllInfo object

bondforcefene.setParams('polymer', 10, 1.2)

#sets parameters by: bond type, spring constant, maximum length of the bond

app.add(bondforcefene)

#adds this object to the application

6.3.3 Bond polynomial force

Constructor:

BondForcePolynomial (boost::shared_ptr<AllInfo>)

#note: specifies the polynomial bond force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real, Real)

#note: specifies the polynomial bond force parameters

#parameters: bond type, spring constant k_1, spring constant k_2, equilibrium bond length

Example:

bondforcepolynomial = galamost. BondForcePolynomial (all_info)

#initializes BondForceFene object by: AllInfo object

bondforcepolynomial.setParams('polymer', 10.0, 100.0, 1.2)

#sets parameters by: bond type, spring constant, spring constant, equilibrium bond length

app.add(bondforcepolynomial)

#adds this object to the application

6.3.4 Angle harmonic force

Constructor:

AngleForceHarmonic(boost::shared_ptr<AllInfo>)

#note: specifies the angle harmonic force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real)

#note: specifies the angle harmonic force parameters

#parameters: angle type, spring constant, equilibrium angle degree

Example:

angleforce = galamost.AngleForceHarmonic(all_info)

#initializes AngleForceHarmonic object by: AllInfo object

angleforce.setParams('P-G-G', 25.000, 120.000)

#sets parameters by: angle type, spring constant, equilibrium degree

app.add(angleforce)

#adds this object to the application

6.3.5 Angle cosine harmonic force

Constructor:

AngleForceHarmonicCos (boost::shared_ptr<AllInfo>)

#note: specifies the angle cosine harmonic force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real)

#note: specifies the angle cosine harmonic force parameters

#parameters: angle type, spring constant, equilibrium angle degree

Example:

```
angleforce = galamost.AngleForceHarmonicCos(all_info)
#initializes AngleForceHarmonic object by: AllInfo object
angleforce.setParams('P-G-G', 25.000, 120.000)
#sets parameters by: angle type, spring constant, equilibrium degree
app.add(angleforce)
#adds this object to the application
```

6.3.6 Angle cosine force

Constructor:

AngleForceCos (boost::shared_ptr<AllInfo>)

#note: specifies the angle cosine force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real)

#note: specifies the angle cosine force parameters

#parameters: angle type, spring constant, equilibrium angle degree

Example:

```
angleforce = galamost.AngleForceCos(all_info)
#initializes AngleForceCos object by: AllInfo object
angleforce.setParams('P-G-G', 25.000, 120.000)
#sets parameters by: angle type, spring constant, equilibrium degree
app.add(angleforce)
#adds this object to the application
```

6.3.7 Dihedral harmonic force

Constructor:

DihedralForceHarmonic(boost::shared_ptr<AllInfo>)

#note: specifies the dihedral harmonic force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real)

#note: specifies the dihedral harmonic force parameters

#parameters: dihedral type, spring constant, equilibrium dihedral degree

Example:

dihedralforce = galamost. DihedralForceHarmonic (all_info)

#initializes DihedralForceHarmonic object by: AllInfo object

dihedralforce.setParams('A-B-B-A', 10.0, 0.0)

#sets parameters by: dihedral type, spring constant, equilibrium degree

app.add(dihedralforce)

#adds this object to the application

6.3.8 Dihedral Opls Cosine force

Constructor:

DihedralForceOplsCosine (boost::shared_ptr<AllInfo>)

#note: specifies the dihedral opls cosine force

#parameter: AllInfo

Functions:

void setParams(const std::string &, Real, Real, Real, Real, Real)

#note: specifies the dihedral opls cosine force parameters

#parameters: dihedral type, C_0, C_1, C_2, C_3, equilibrium dihedral degree

Example:

dihedralforce = galamost.DihedralForceOplsCosine(all_info)

#initializes DihedralForceOplsCosine object by: AllInfo object

dihedralforce.setParams('C_33-C_32-C_32-C_32', 0.0, 2.95188, -0.566963, 6.57940, 0.0)

#sets parameters by: dihedral type, C_0, C_1, C_2, C_3, equilibrium dihedral degree

app.add(dihedralforce)

#adds this object to the application

6.4 Integration method

6.4.1 Nve ensemble

Constructor:

Nve(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>)

#note: specifies the NVE thermostat of a group of particles

#parameter: AllInfo, ParticleSet

Functions:

void setZeroForce(bool)

#note: switches the function of making all force to be zero

#parameters: True or False (the default is False)

Example:

thermo = galamost.Nve(all_info, group)

#initializes Nve object by: AllInfo object, ParticleSet object

app.add(thermo)

#adds this object to the application

6.4.2 Nvt ensemble with Nose Hoover method

Constructor:

NoseHooverNvt(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>,

boost::shared_ptr<ComputeInfo>, Real, Real)

#note: specifies the NVT NoseHoover thermostat of a group of particles

#parameter: AllInfo, ParticleSet, ComputeInfo, temperature, tauT

Functions:

void setT(Real)

#note: specifies the temperature

#parameters: a fixed value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a defined varing temperature according to time steps

Example:

nh = galamost.NoseHooverNvt(all_info, group, comp_info, 1.0, 0.5)

#initializes NoseHooverNvt object by: AllInfo object, ParticleSet object, ComputeInfo object, temperature, tauT

app.add(nh)

#adds this object to the application

6.4.3 Nvt ensemble with Berendsen method

Constructor:

BerendsenNvt (boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>, boost::shared_ptr<ComputeInfo>, Real, Real)

#note: specifies the NVT Berendsen thermostat of a group of particles

#parameter: AllInfo, ParticleSet, ComputeInfo, temperature, tauT

Functions:

void setT(Real)

#note: specifies the temperature

#parameters: a fixed value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a defined varing temperature according to time steps

6.4.4 Nvt ensemble with Andersen method

Constructor:

AndersenNvt (boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>, Real, Real, unsigned int)

#note: specifies the NVT Andersen thermostat of a group of particles

#parameter: AllInfo, ParticleSet, temperature, collision frequency, RNG seed

Functions:

void setT(Real)

#note: specifies the temperature

#parameters: a fixed value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a defined varing temperature according to time steps

Example:

an = galamost. AndersenNvt (all_info, group, 1.0, 10.0, 12345)

#initializes AndersenNvt object by: AllInfo object, ParticleSet object, temperature, collision frequency, RNG seed

app.add(an)

#adds this object to the application

6.4.5 Npt ensemble with Andersen barostat

Constructor:

Npt(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>, boost::shared_ptr<ComputeInfo>, boost::shared_ptr<ComputeInfo>, Real, Real, Real, Real)

#note: specifies the NPT thermostat of a group of particles

#parameter: AllInfo, ParticleSet, ComputeInfo, ComputeInfo, temperature, pressure, tauT, tauP

Functions:

void setP(Real)

#note: specifies the pressure

#parameters: a fixed value of pressure

void setT(Real)

#note: specifies the temperature

#parameters: a fixed value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a defined varing temperature according to time steps

Example:

npt = galamost.Npt(all_info, group, comp_info, comp_info, 1.0, 0.2, 0.5, 0.1)

#initializes Npt object by: AllInfo object, ParticleSet object, ComputeInfo

object for the group, ComputeInfo object for all particles, temperature, pressure, tauT, tauP)

app.add(npt)

#adds this object to the application

6.4.6 Nve rigid body thermostat

Constructor:

NveRigid(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>)

#note: specifies the NVE thermostat of rigid bodies

#parameter: AllInfo, ParticleSet

Example:

rigidnve = galamost.NveRigid(all_info, bgroup)

#initializes NveRigid object by: AllInfo object, ParticleSet object

app.add(rigidnve)

#adds this object to the application

6.4.7 Nvt rigid body thermostat

Constructor:

NveRigid(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>)

#note: specifies the NVE thermostat of rigid bodies

#parameter: AllInfo, ParticleSet

Functions:

void setT(Real)

#note: specifies the temperature

#parameters: a fixed value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a defined varing temperature according to time steps

Example:

rigidnvt = galamost.NvtRigid(all_info, bgroup, 1.0, 10.0)

#initializes NvtRigid object by: AllInfo object, ParticleSet object, temperature, tau

app.add(rigidnvt)

#adds this object to the application

6.4.8 Brownian dynamic nvt rigid body thermostat

Constructor:

BdNvtRigid(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>, Real, unsigned int)

#note: specifies the Brownian NVT thermostat of rigid bodies

#parameter: AllInfo, ParticleSet, temperature RNG seed

Functions:

void setGamma(Real)

#note: specifies the gamma of Brownian method

#parameters: a fixed value of gamma

void setGamma(const std::string &, Real)

#note: specifies the gamma of Brownian method of different types

#parameters: type, gamma

void setT(Real)

#note: specifies the temperature

#parameters: a fixed value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a defined varing temperature according to time steps

Example:

bdrigidnvt = galamost. BdNvtRigid(all_info, bgroup, 1.0, 123)

#initializes BdNvtRigid object by (AllInfo object, ParticleSet object, temperature, RNG seed)

app.add(bdrigidnvt)

#adds this object to the application

6.4.9 Nve rigid body thermostat with tunable freedoms

Constructor:

TranRigid(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>)

#note: specifies the NVE thermostat of rigid bodies in defined freedoms

#parameter: AllInfo, ParticleSet

Functions:

void setTraDimention(bool, bool, bool)

#note: switches the freedoms of translocation in x y z directions

#parameters: True or False for x y z translocation, respectively

void setRotDimention(bool, bool, bool)

#note: switches the freedoms of rotation in x y z directions

#parameters: True or False for x y z rotation, respectively

Example:

```
bdrigidnvt = galamost.TranRigid (all_info, bgroup)
#initializes TranRigid object by: AllInfo object, ParticleSet object
bdrigidnvt.setTraDimention(True, True, True)
#switches the freedoms of translocation in x y z directions
bdrigidnvt.setRotDimention(True, True, True)
#switches the freedoms of rotation in x y z directions
app.add(bdrigidnvt)
#adds this object to the application
```

6.5 Particle Group

6.5.1 Build group by keywords

Constructor:

ParticleSet(boost::shared_ptr<AllInfo>, const std::string&)

#note: specifies the group of particles by keywords

#parameter: AllInfo, keywords

Example:

```
groupC = galamost.ParticleSet(all_info, 'C')
#initializes ParticleSet object by: AllInfo object, particle type
groupB = galamost.ParticleSet(all_info, 'body')
#initializes ParticleSet object of body particles by: AllInfo object, key word
groupNB= galamost.ParticleSet(all_info, 'non_body')
```

```
#initializes ParticleSet object of nonbody particles by: AllInfo object, key
word
groupC = galamost.ParticleSet(all_info, 'all')
#initializes ParticleSet object of all particles by: AllInfo, key word
```

6.5.2 Build group by index range of particles

Constructor:

```
ParticleSet(boost::shared_ptr<AllInfo>, unsigned int, unsigned int)
#note: specifies the group of particles in a range of indexes
#parameter: AllInfo, minimum index, maximum index
```

Example:

```
groupC = galamost.ParticleSet(all_info, 0, 10)
#initializes ParticleSet object by: AllInfo object, min particle index, max
particle index
```

6.5.3 Combine groups

Functions:

```
boost::shared_ptr<ParticleSet> combine(boost::shared_ptr<ParticleSet> ,
boost::shared_ptr<ParticleSet> )
#note: combines two groups into one
#parameter: ParticleSet, ParticleSet
```

Example:

```
groupAB= galamost.ParticleSet. combine (groupA, groupB)
#initializes ParticleSet object by: ParticleSet object, ParticleSet object
```

6.6 Particle List

6.6.1 Build neighbor list

Constructor:

```
NeighborList(boost::shared_ptr<AllInfo>, Real, Real)
#note: specifies the neighbor list of all particles
#parameters: AllInfo, cut-off, buffer distance
```

Functions:

void setRCut(Real, Real)

#note: specifies the cut-off and buffer distance

#parameters: cut-off, buffer distance

void setNsq()

#note: switches on the method of searching all particles to build up list

void setDataReproducibility()

#note: switches on the data reproducibility

Example:

neighbor_list = galamost.NeighborList(all_info, 3.0 ,0.4)

#initializes NeighborList object by: AllInfo object, cut off, buffer distance

6.6.2 Build cell list

Constructor:

CellList(boost::shared_ptr<AllInfo>)

#note: specifies the cell list of all particles

#parameter: AllInfo

Functions:

void setNominalWidth (Real)

#note: specifies the length of cell

#parameters: the length of cell

void setNominalDim(unsigned int, unsigned int, unsigned int)

#note: specifies the dimentions of grid in x, y, z directions, respectively

#parameters: the number of cells in x, y, z directions

void setDataReproducibility()

#note: switches on the data reproducibility function

Example:

cell_list = galamost.CellList(all_info);

#initializes CellList object by: AllInfo object

6.7 Data dump

6.7.1 The dump of some important informations

Constructor:

DumpInfo(boost::shared_ptr<AllInfo> , boost::shared_ptr<ComputeInfo>, const std::string&)

#note: specifies the information dump method of a group of particles

#parameters: AllInfo, ComputeInfo, file name

Functions:

void dumpAnisotropy()

#note: switches on the dump of information related to anisotropic particles

void dumpVirialEnergy(boost::shared_ptr<Force>)

#note: switches on the dump of energy and virial of inputting force object

#parameters: Force object

Example:

```
dInfo = galamost.DumpInfo(all_info, comp_info, 'data.log')
#initializes DumpInfo object by: AllInfo object, ComputeInfo object, file name
dInfo.setPeriod(200)
#sets dump period
app.add(dInfo)
#adds this object to the application
```

6.7.2 Configuration dump with mol2 format

Constructor:

Mol2Dump(boost::shared_ptr<AllInfo>, std::string)

#note: specifies the dump method of mol2 files

#parameters: AllInfo, file name

Functions:

void setChangeFreeType(const std::string&)

#note: specifies the type of free particles which will be changed to be 'F'

void deleteBoundaryBond(bool)

#note: switches on the function of undisplaying the bonds across the box

Example:

```
mol2 = galamost.Mol2Dump(all_info, 'particles')
#initializes Mol2Dump object by: AllInfo object, base file name
mol2.setPeriod(100000)
#sets dump period
mol2.deleteBoundaryBond(True)
#undisplays the bonds which across the box
app.add(mol2)
#adds this object to the application
```

6.7.3 Configuration dump with xml format

Constructor:

XmlDump(boost::shared_ptr<AllInfo>, std::string)

#note: specifies the dump method of XML files

#parameters: AllInfo, base file name

Functions:

void setOutputPosition(bool)

#note: switches the function of outputing position (the default is true)

void setOutputType (bool)

#note: switches the function of outputing type (the default is true)

void setOutputImage (bool)

#note: switches the function of outputing image

void setOutputVelocity (bool)

#note: switches the function of outputing velocity

void setOutputMass (bool)

#note: switches the function of outputing mass

void setOutputCharge (bool)

#note: switches the function of outputing charge

void setOutputDiameter (bool)

#note: switches the function of outputing diameter

void setOutputBody (bool)

#note: switches the function of outputing body

void setOutputVirial (bool)

#note: switches the function of outputing virial

void setOutputForce (bool)

#note: switches the function of outputing force

void setOutputOrientation (bool)

#note: switches the function of outputing orientation

void setOutputQuaternion (bool)

#note: switches the function of outputing quaternion

void setOutputRotation (bool)

#note: switches the function of outputing rotation velocity

void setOutputTorque (bool)

#note: switches the function of outputing torque

void setOutputInert (bool)

#note: switches the function of outputing inert tensor

void setOutputInit (bool)

#note: switches the function of outputing initiator indication

void setOutputCris (bool)

#note: switches the function of outputing cris

void setOutputBond (bool)

#note: switches the function of outputing bond

void setOutputAngle (bool)

#note: switches the function of outputing angle

void setOutputDihedral (bool)

#note: switches the function of outputing dihedral

Example:

```
xml = galamost.XmlDump(all_info, 'particles')
#initializes XmlDump by: AllInfo object, filename
xml.setPeriod(100000)
#sets dump period
```

app.add(xml)

#adds this object to the application

6.7.4 Configuration dump with dcd format

Constructor:

DcdDump(boost::shared_ptr<AllInfo>, const std::string &, bool)

#note: specifies the dump method of DCD file

#parameters: AllInfo, file name, overwrite former data

Functions:

void removePbc(bool)

#note: switches the function of unwrapping the molecules folded by PBC

Example:

dcd = galamost.DcdDump(all_info, 'particles',True)

#initializes XmlDump object by: AllInfo object, filename, overwriting

dcd.removePbc(True)

#unwraps the molecules folded by PBC

dcd. DcdDump (100000)

#sets dump period

app.add(dcd)

#adds this object to the application

6.7.5 Configuration dump with binary format

Constructor:

DcdDump(boost::shared_ptr<AllInfo>, const std::string &, bool)

#note: specifies the dump method of DCD file

#parameters: AllInfo, file name, overwrite former data

Functions:

void setOutputAll()

#note: switches on the function of outputing all data

void setOutputForRestart()

#note: switches on the function of outputing data needed for restarting

void enableCompression(bool)

#note: switches the function of compressing output file

Example:

binary = galamost.BinaryDump(all_info, 'particle')

#initializes BinaryDump object by: AllInfo object, filename

binary.setPeriod(10000)

#sets dump period

app.add(binary)

#adds this object to the application

6.8 Some other functions

6.8.1 Removing the momentum of center of mass

Constructor:

ZeroMomentum(boost::shared_ptr<AllInfo>)

#note: specifies the method of making the momentum of center mass of all particles to be zero

#parameter: AllInfo

ZeroMomentum(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>)

#note: specifies the method of making the momentum of center mass of a group of particles to be zero

Example:

zm = galamost.ZeroMomentum(all_info)

#initializes ZeroMomentum object by: AllInfo object

zm.setPeriod(10)

#sets computing period

app.add(zm)

#adds this object to the application

6.8.2 The computition of some important informations

Constructor:

ComputeInfo(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>)

#note: specifies the method of dumping some important information

#parameters: AllInfo, ParticleSet

Functions:

void setNdof(unsigned int)

#note: sets the freedom degree

Example:

comp_info = galamost.ComputeInfo(all_info, group)

#initializes ComputeInfo object by: AllInfo object, ParticleSet object

6.8.3 Sorting data

Constructor:

ComputeInfo(boost::shared_ptr<AllInfo>, boost::shared_ptr<ParticleSet>)

#note: specifies the method of dumping some important information

#parameters: AllInfo, ParticleSet

Example:

sort_method = galamost.Sort(all_info)

#initializes Sort object by: AllInfo object

sort_method.setPeriod(300)

#sets computing period

app.add(sort_method)

#adds this object to the application

6.8.4 VariantLinear

Constructor:

VariantLinear ()

#note: specifies linearly varing method

Functions:

void setPoint(unsigned int, double)

#note: specifies the value at the time step

#parameters: time step, value

Example:

v = galamost.VariantLinear()

#initializes VariantLinear object

v. setPoint (0, 1.0)

v. setPoint (100000, 2.0)

#set the value at the time step and the value at any time step can be gotten by linear interpolation.

6.8.5 VariantSin

Constructor:

VariantSin ()

#note: specifies sinusoidal curve varing method

Functions:

void setPoint(unsigned int, double, double, double)

#note: specifies the period, upper, and lower bounds at the time step

#parameters: time step, period, upper, and lower bounds

Example:

v = galamost.VariantSin ()

#initializes VariantSin object

v. setPoint (0, 1000, 1.0, -1.0)

v. setPoint (100000, 1000, 2.0, -2.0)

#set the parameters of sinusoid at the time step and the parameters at any time step can be gotten by linear interpolation.

6.8.6 VariantWell

Constructor:

VariantWell()

#note: specifies well curve varing method

Functions:

void setPoint(unsigned int, double, double, double)

#note: specifies the period, upper, and lower bounds at the time step

#parameters: time step, period, upper, and lower bounds

Example:

v = galamost.VariantWell ()

#initializes VariantWell object

v. setPoint (0, 1000, 1.0, -1.0)

v. setPoint (100000, 1000, 1.0, -1.0)

#set the parameters of periodic well at the time step and the parameters at any time step can be gotten by linear interpolation.

6.9 Some functional modules

6.9.1 The coarse-graining numerical method

1) Non-bonded interaction:

Constructor:

PairForceTable(boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>, unsigned int)

#note: specifies the method of numerical pair force

#parameters: AllInfo, NeighborList, the number of potential points

Functions:

void setParams(const std::string &, const std::string &, Real, const std::string &, int, int)

#note: specifies the numerical pair interaction parameters by pair types

#parameters: type 1, type 2, cut-off, inputting file name, start column, end column

Example:

pair = galamost.PairForceTable(all_info, neighbor_list, 1.3, 2000)

#initializes PairForceTable object by: AllInfo object, NeighborList object, cut off, the number of inputting data

pair.setParams('A', 'A' ,1.3, "table.dat", 0, 3)

#sets parameters by: type 1, type 2, cut-off, inputting file name, start column, end column (four columns for C_1 C_2 C_3 C_4)

app.add(pair)

#adds this object to the application

2) Bond interaction:

Constructor:

BondForceTable(boost::shared_ptr<AllInfo>, Real, unsigned int)

#note: specifies the method of numerical bond force

#parameters: AllInfo, cut-off, the number of potential points

Functions:

void setParams(const std::string &, const std::string &, int ,int)

#note: specifies the numerical bond interaction parameters by bond types

#parameters: bond type, inputting file name, start column, end column

Example:

```
bond = galamost.BondForceTable(all_info, 2.0, 2000)
#initializes BondForceTable object by :AllInfo object, cut off, data number
bond.setParams('1_1' , "table.dat", 0, 3)
#sets parameters by: bond type, cut-off, inputting file, start column, end
column
app.add(bond)
#adds this object to the application
```

3) Angle interaction:

Constructor:

AngleForceTable(boost::shared_ptr<AllInfo>, unsigned int)

#note: specifies the method of numerical angle force

#parameters: AllInfo, the number of potential points

Functions:

void setParams(const std::string &, const std::string &, int ,int)

#note: specifies the numerical angle force parameters by bond types

#parameters: angle type, inputting file name, start column, end column

Example:

```
angle = galamost.AngleForceTable(all_info, 500)
#initializes AngleForceTable object by: AllInfo object, data number
angle.setParams('111' , "table.dat", 0, 3)
#sets parameters by: angle type, inputting file, start column, end column
app.add(angle)
#adds this object to the application
```

4) Dihedral interaction

Constructor:

DihedralForceTable (boost::shared_ptr<AllInfo>, unsigned int)

#note: specifies the method of numerical dihedral force

#parameters: AllInfo, the number of potential points

Functions:

void setParams(const std::string &, const std::string &, int ,int)

#note: specifies the numerical dihedral force parameters by dihedral types

#parameters: dihedral type, inputting file name, start column, end column

Example:

dihedral = galamost. DihedralForceTable (all_info, 500)

#initializes DihedralForceTable object by: AllInfo object, data number

dihedral.setParams('111' , "table.dat", 0, 3)

#sets parameters by: dihedral type, inputting file, start column, end column

app.add(dihedral)

#adds this object to the application

6.9.2 The MD-SCF method

Constructor:

MdScfForce(boost::shared_ptr<AllInfo>, unsigned int , unsigned int, unsigned int, Real)

#note: specifies the method of MD-SCF force

#parameters: AllInfo, grid number in x, y, z, direction, compressibility

Functions:

void setParams(const std::string &, const std::string &, Real)

#note: specifies the MD-SCF interaction parameters by pair types

#parameters: type 1, type 2, χ parameter

void setNewVersion(bool)

#note: switches the function of newly developed method of implementation

Example:

scf = galamost.MdScfForce(all_info, 22, 22, 22, 0.100)

#initializes MdScfForce object by: AllInfo object, grid number in x, y, z directions, compressibility

scf.setParams('N', 'N', 0.000)

#sets parameters by: type1, type 2, χ parameters

scf.setParams('N', 'P', -1.500)

#sets parameters by: type1, type 2, χ parameters

scf.setParams('P', 'P', 0.000)

#sets parameters by: type1, type 2, χ parameters

scf.setPeriodScf(1, 300)

#sets parameters by: computing period of density field, updating period of density field

scf.setNewVersion(True)

#switches to newly developed version

app.add(scf)

#adds this object to the application

6.9.3 The Polymerization method

Constructor:

Polymerization(boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>, Real, unsigned int)

#note: specifies the method of polymerization

#parameters: AllInfo, NeighborList, cut-off, RNG seed

Polymerization(boost::shared_ptr<AllInfo>, const std::string &, Real, boost::shared_ptr<NeighborList>, Real, unsigned int)

#note: specifies the method of polymerization with a percent of initiator

#parameters: AllInfo, type, percent, NeighborList, cut-off, RNG seed

Functions:

void setPr(Real)

#note: specifies the reaction probability

void setPr(const std::string&, const std::string&, Real)

#note: specifies the reaction probability between two kind particles

#parameters: particle type 1, particle type 2, probability

void setPrFactor (Real)

#note: specifies the reaction probability factor of factor^{n} where n is the times

the reactive end has reacted

void setPrFactor (const std::string&, const std::string&, Real)

#note: specifies the reaction probability factor between two kind particles

#parameters: particle type 1, particle type 2, probability factor

void setExchangePr(const std::string &, const std::string &, const std::string &, Real)

#note: specifies the reaction probability of replacing particle 3 to connect to paritlce 2 by particle1

#parameters: particle type 1, particle type 2, particle type 3, probability

void setMaxCris(const std::string &, unsigned int)

#note: specifies the upper limit number of bonds generated by reaction

#parameters: particle type, upper limit number

void setNewBondType(const std::string &)

#note: specifies the new bond type generated by reaction

#parameters: bond type

void setFuncReactRule(bool, Real, Real, Real, Real, Func)

#note: specifies the reaction probability according to bond potential

#parameters: enable or disenable this method, spring constant, equilibrium length for harmonic interaction or maximum length for FENE interaction, the energy to shift, bond potential type (harmonic or FENE)

void setMinDisReactRule(bool)

#note: switches the reaction rule of nearest reaction or near reaction

#parameters: True for nearest reaction

void initExPoint ()

#note: switches on initializing reactive point for exchange reaction

Example:

reaction = galamost.Polymerization(all_info, neighbor_list, 1.12246 ,16361)

#initializes Polymerization object by: AllInfo object, NeighborList object, cut-off, RNG seed

reaction.setFuncReactRule(True, 1250.000, 1.0, 0.470, 10.0, galamost.Polymerization.Func.harmonic)

#sets the bond connection method after reaction and conveys parameters by: function rule, k0, r0, b0, epsilon, function type

reaction.setPr(0.002)

#sets the reaction probability

reaction.setMaxCris('B',3)

#sets the connected bond uplimit number

reaction.setPeriod(50)

#sets the reaction period

app.add(reaction)

#adds this object to the application

6.9.4 Anisotropic model

1) Force and torque:

Constructor:

LzwForce(boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>, Real)

#note: specifies the method of LZW force

#parameters: AllInfo, NeighborList, cut-off

Functions:

void setParams(const std::string &, const std::string &, Real, Real, Real, Real, Real)

#note: specifies the LZW force parameters by pair types

#parameters: type 1, type 2, alphaR, mu, nu, alphaA, beta

void setMethod(const std::string &)

#note: choses a method of 'Disk', 'Janus', ABAtriJanus', 'BABtriJanus'

Example:

lzw = galamost.LzwForce(all_info, neighbor_list, 1.0)

#initializes LzwForce object by: AllInfo object, NeighborList object, cut-off

lzw.setParams('A', 'A' , 396.0, 1.0, 0.5, 88.0, 60.0/180.0*3.1415926)

#sets parameters by: type1, type 2, alphaR, mu, nu, alphaA, beta

lzw.setMethod('ABAtriJanus')

#sets method with the choise of ABAtriJanus

app.add(lzw)

#adds this object to the application

2) Thermostat:

Constructor:

BerendsenAniNvt(boost::shared_ptr<AllInfo>,boost::shared_ptr<ParticleSet>, boost::shared_ptr<ComputeInfo>, Real, Real, Real)

#note: specifies the Berendsen NVT thermostat for anisotropic particles

#parameters: AllInfo, ParticleSet, ComputeInfo, temperature, tauT, tauR

Functions:

void setTau(Real, Real)

#note: specifies the Berendsen NVT thermostat

#parameters: tauT, tauR

void setT(Real)

#note: specifies the temperature

#parameters: a fixed value of temperature

void setT(boost::shared_ptr<Variant>)

#note: specifies the temperature

#parameters: a defined varing temperature according to time steps

Example:

bere = galamost.BerendsenAniNvt(all_info, group, comp_info, 1.0, 0.3, 0.1)

#initializes BerendsenAniNvt object by: AllInfo object, NeighborList object, ParticleSet object, ComputeInfo object, temperature, tauT, tauR

app.add(bere)

#adds this object to the application

6.9.5 Gay-Berne model

1) GB Pair Force

Constructor:

GBForce (boost::shared_ptr<AllInfo>, boost::shared_ptr<NeighborList>, Real)

#note: specifies the method of Gay-Berne force

#parameters: AllInfo, NeighborList, cut-off

Functions:

void setParams(const std::string &, const std::string &, Real, Real, Real, Real, Real, Real, Real, Real, Real)

#note: specifies the GB force parameters by pair types

#parameters: type1, type 2, epsilon0, sigma0, nu, mu, end-to-end length (sigmae), side-by-side length (sigmas), end-to-end energy (epsilone), side-by-side energy (epsilons), Ps

Example:

```
gb = galamost.GBForce(all_info, neighbor_list, 10.0)
#initializes GBForce object by: AllInfo object, NeighborList object, cut-off
gb.setParams('A', 'A' , 1.5, 1.5, 1.0, 2.0, 3.0, 1.0, 0.5, 3.0, 1.0)
#sets parameters by: type 1, type 2, epsilon0, sigma0, nu, mu, sigmae, sigmas, epsilone, epsilons, Ps
app.add(gb)
#adds this object to the application
```

2) Bond Force of anisotropic particles

Constructor:

BondForceAni(boost::shared_ptr<AllInfo>)

#note: specifies the method of bond force of anisotropic particles

#parameters: AllInfo

Functions:

void setParams(const std::string &, Real, Real, Real, Real)

#note: specifies the bond force parameters by bond types

#parameters: bond type, bond spring constant, end-to-end length of GB particle, angle spring constant, equilibrium angle degree

Example:

```
bondani = galamost.BondForceAni(all_info)
#initializes BondForceAni object by: AllInfo object
bondani.setParams('A-A', 100.0 , 4.498, 30.0, 0.0)
#sets parameters by: bond type, bond spring constant, sigmae, angle spring constant, equilibrium angle degree
```

app.add(bondani)

#adds this object to the application

6.10 Configuration generator

1) The head of script:

#!/usr/bin/python

import sys

sys.path.append('/opt/galamost3/lib')

import molgen

#sets the path of molgen.so and imports extended Python module of molgen

2) Molecules defination:

Constructor:

Molecule(unsigned int)

#note: defines a molecule by Molecule

#parameters: the number of particles in a molecule

Molecule(const std::string&, unsigned int)

note: defines a molecule by Molecule and reads partial data from a file

#parameters: file name, number of particles in a molecule

Functions:

void setParticleTypes(std::string)

#note:specifies the particle types seperated by comma with particle index form 0 to N-1 in sequence

void setTopology(std::string)

#note: specifies bonds seperated by comma which connect two particles seperated by crossband

void setIsotactic(bool)

#note: switches the isotactic configuration of molecule

void setBondLength(double)

#note: specifies the bond length of all bonds

void setBondLength(std::string, std::string, double)

#note: specifies the bond length of the bond which connect two kind particles

#parameters: particle type 1, particle type2, bond length

void setAngleDegree(std::string, std::string, std::string, double)

#note: specifies the angle degree of the angle consisting of three kind particles

#parameters: particle type 1, particle type2, particle type3, angle degree

void setAngleDegree(unsigned int, unsigned int, unsigned int, double)

#note: specifies the angle degree of the angle consisting of three particles

#parameters: particle 1, particle 2, particle 3, angle degree

void setDihedralDegree (std::string, std::string, std::string, std::string, double)

#note: specifies the dihedral degree of the dihedral consisting of four kind particles

#parameters: particle type 1, particle type2, particle type3, particle type4, dihedral degree

void setDihedralDegree (unsigned int, unsigned int, unsigned int, unsigned int, double)

#note: specifies the dihedral degree of the dihedral consisting of four particles

#parameters: particle 1, particle 2, particle 3, particle 4, dihedral degree

void setMass (double)

#note: specifies the mass of all kind particles

#parameters: mass

void setMass (std::string, double)

#note: specifies the mass of the kind particle

#parameters: particle type, mass

void setMass (unsigned int, double)

#note: specifies the mass of the particle

#parameters: particle index, mass

void setCharge (double)

#note: specifies the charge of all kind particles

#parameters: charge

void setCharge (std::string, double)

#note: specifies the charge of the kind particle

#parameters: particle type, charge

void setCharge (unsigned int, double)

#note: specifies the charge of the particle

#parameters: particle index, charge

void setOrientation (double)

#note: specifies the orientation of all kind particles

#parameters: orientation

void setOrientation (std::string, double)

#note: specifies the orientation of the kind particle

#parameters: particle type, orientation

void setOrientation (unsigned int, double)

#note: specifies the orientation of the particle

#parameters: particle index, orientation

void setDiameter (double)

#note: specifies the diameter of all kind particles

#parameters: diameter

void setDiameter (std::string, double)

#note: specifies the diameter of the kind particle

#parameters: particle type, diameter

void setDiameter (unsigned int, double)

#note: specifies the diameter of the particle

#parameters: particle index, diameter

void setCris (unsigned int)

#note: specifies the cris of all kind particles

#parameters: cris

void setCris (std::string, unsigned int)

#note: specifies the cris of the kind particle

#parameters: particle type, cris

void setCris (unsigned int, unsigned int)

#note: specifies the cris of the particle

#parameters: particle index, cris

void setInit (unsigned int)

#note: specifies the init of all kind particles

#parameters: init

void setInit (std::string, unsigned int)

#note: specifies the init of the kind particle

#parameters: particle type, init

void setInit (unsigned int, unsigned int)

#note: specifies the init of the particle

#parameters: particle index, init

Example:

```
mol0=molgen.Molecule(8)
#initializes Molecule object by: the number of particles
mol0.setParticleTypes("A,A,A,A,A,A,A,A")
#sets particle types
mol0.setTopology("0-1,0-3,0-4,2-3,1-2,1-5,2-6,3-7,4-5,4-7,5-6,6-7")
#sets topology
mol0.setBondLength(0.75)
#sets bond length for all bonds
mol0.setMass(1.0)
#sets mass for all particle
mol0.setAngleDegree("A","A","A",90.0)
#sets the degree of the angle of particles with the type 1, 2 and 3
```

3) Objects defination:

Constructor:

Object (unsigned int, Object::Shape)

#note: defines a object by Object

#parameters: the number of particles in a molecule, shape

Object (const std::string&, unsigned int, Object::Shape)

note: defines a object by Object and reads partial data from a file

#parameters: file name, number of particles in a molecule, shape

Functions:

void setBody ()

#note: specifies all particles in the molecule to be in the body

#parameters: no

void setBody (std::string)

#note: specifies the kind particle to be in the body

#parameters: particle type

void setBody (unsigned int)

#note: specifies the particle to be in the body

#parameters: particle index

void setRadius (double)

#note: specifies the radius of the sphere which will be generated

#parameters: radius

Example:

```
mol0 = molgen.Object("sphere2.xml", 65, molgen.Object.Shape.sphere)
#initializes Object object by: the reading file, the number of particles, shape
mol0.setParticleTypes("C*60,A*5")
#sets particle types
mol0.setTopology("59-60,60-61,61-62,62-63,63-64")
#sets topology
mol0.setBody("C");
#sets body particles
```

4) Generator defination:

Constructor:

Generators(double, double, double)

#note:specifies a generator by Generators

#parameters: box length in x y z directions

Functions:

void addMolecule(boost::shared_ptr<Molecule>, unsigned int)

#note:adds molecules into generator

#parameters: Molecule, the number of molecules

void setMinimumDistance (double)

#note: sets minimum seperated distance of all particles

void setMinimumDistance (const std::string&, const std::string&, double)

#note: sets minimum seperated distance of two kind particles

#parameters: particle type 1, particle type 2, minimum distance

void setParam(const string&, const string&, double, double, double)

#note: sets LJ potential parameters between two kind particles for Rosenblueth method

#parameters: particle type 1, particle type 2, epsilon, sigma, cut-off

void outPutXml(std::string)

#note: switch on the function of output of XML files

#parameters: XML filename

void outPutMol2 (std::string)

#note: switch on the function of output of Mol2 files

#parameters: Mol2 filename

Example:

```
gen=molgen.Generators(10,10,10)
#initializes Generators object by: box length in x, y, and z direction
gen.addMolecule(mol0,10)
#adds Molecule by: Molecule name, the number of molecules
gen.setParam("A","A", 1.0, 0.7, 1.0)
#sets parameters of LJ potential which is used for Rosenblueth method
gen.setMinimumDistance(0.7);
#sets minimum seperated distance of all particles
gen.setMinimumDistance("A","A",0.7);
#sets minimum seperated distance between the particle types
gen.outPutXml("test");
#sets file name of output XML file
```

6.11 Data tackle

The "galaTackle" is a tool to get some important information by analyzing the output raw data from configuration files. You may call this tool by running command:

galaTackle particle.xml

where "galaTackle" can be found in the installed "bin" file and the XML configuration files should be put in current directory.

After the executation of above command, a menu of many options will be listed. You can chose one or several functions by the number index seperated by blank and press enter to execute.

ID	Functions	Paramters	Description
1	Rg^2		mean square of gyration radius
2	Ed^2		mean square of end-end distance
3	RDF	:maxbin=100|gpu_id=0	radial distribution function
4	bond_distri	:npot=2001	bond length distribution
5	angle_distri	:npot=2001	angle degree distribution
6	dihedral_distri	:npot=2001	dihedral degree distribution
7	stress tensor	:bondex=true|bodyex=false|rcut=3.0	stress tensor
8	density		density in real unit
9	re-cal-image		removes PBC and keeps the center mass in box
10	MSD		mean square displacement of all particles
11	RDF_{CM}	:maxbin=100|gpu_id=0	the RDF of center mass of molecules
12	MSD_{CM}		the MSD of center mass of molecules

13	ents		analyzing entanglement information of long chains
14	strfac	:kmax=80|gpu_id=0	structure factor
15	domain size	:kmax=20|qc=0.4690|gpu_id=0	domain size
16	dynamic strfac	:kmax=20|q=7.0	dynamic structure factor
17	config check	:bondex=true|bodyex=false|rcut=2.0	checks the configuration from XML file

7 Appended Documents

7.1 Polymerization model

可反应的类型：

1. 自由基聚合反应
   通式：AAAA*+A➔AAAAA*
   特点：反应有方向性，从自由基出发，将活性位点向前移动
   程序中的反应模型：自由基聚合反应
   例：
   reaction =galamost.Polymerization()
   reaction.setPr('A', 'A', 0.005)
   reaction.setPeriod(50)
   app.add(reaction)
   注：这里不能设定 reaction.setMaxCris('A', 2)，否则会改变反应模式，活性位点就无法向前移动。反应具有方向性，比如 reaction.setPr('A', 'B', 0.005)，代表 A 是自由基时（h_init=1）和 B 反应的几率是 0.005，而 B 和 A 的反应几率还是 0。反应几率和初始自由基要匹配，比如 reaction.setPr('A', 'B', 0.005)，若初始只有 B 是自由基时，这种情况不能够反应。setPr(0.005)代表任意对类型间的反应几率都是 0.005.

2. grafting-to 类型反应
   通式：I + ABBBB ➔ I-ABBB
   特点：只反应一次，例如球表面的 I 位点与链 ABBBB 中的 A 只反应一次即结束。因此可利用自由基聚合的模式，但不允许位点迁移。
   程序中的反应模型：逐步聚合反应
   例：
   reaction = galamost.Polymerization(all_info, neighbor_list, 1.2, 16361)
   reaction.setPr('I', 'A', 0.005)
   reaction.setMaxCris('I', 1)
   reaction.setMaxCris('A', 2)
   reaction.setPeriod(200)
   app.add (reaction)
   注：反应具有方向性，比如 reaction.setPr('I', 'A', 0.005)，代表 I 是引发位点（h_init=1）和 A 的反应的几率是 0.005，而 A 和 I 的反应几率还是 0。反应几率和初始自由基要匹配，比如 reaction.setPr('I', 'A', 0.005)，若初始只有 A 是引发位点，这种情况不能够反应。reaction.setMaxCris('I', 1)代表 I 类型粒子的 cris 值为 1 时失去反应活性，同时这个函数也触发了逐步聚合反应模式。
3. 加聚反应：
   通式： AB + CD ➔ AB-CD
   特点：类似 grafting-to 的类型，只反应一次即失活。

程序中的反应模型：逐步聚合反应
例：
Reaction =galamost.Polymerization()
reaction.setPr('B', 'C', 0.005)
reaction.setMaxCris('B', 2)
reaction.setMaxCris('C', 2)
reaction.setPeriod(50)
app.add (reaction)
注：反应具有方向性，上面的体系初始构象 xml 中，设定 h_cris(B)=1, h_cris(C)=1; h_init(B)=1, h_init(C)=0。

4. ligand exchange 反应：
通式：B+ A-C ➔A-B + C
特点：置换过程的存在
程序中的反应模型：交换反应
例
reaction = galamost.Polymerization()
reaction. setExchangePr ('B','A','C',0.1)
app.add (reaction)
体系中有 A-C 键，以 0.1 的几率使 A-C 键断裂，同时 A-B 键成键，首先在 xml 中定义 A 类粒子（或部分 A 类粒子，如 A-C 已成键的 A 类粒子）的 h_init=1,而 B 的 h_cris=0。被置换的 C 粒子的 h_cris>=1，被置换下来后 h_cris 减 1。
5. mobility 反应：
通式：A + C-A➔A-C + A
特点：置换过程的存：
程序中的反应模型：交换反应
reaction = galamost.Polymerization ()
reaction. setExchangePr ('A','C','A',0.1)
app.add (reaction)
体系中有 C-A 键，以 0.1 的几率使 C-A 键断裂，同时 A-C 键成键，首先在 xml 中定义 C 类粒子（或部分 C 类粒子，如 C-A 已成键的 C 类粒子）的 h_init=1,而自由的 A 的 h_cris=0。被置换的 C 粒子的 h_cris>=1，被置换下来后 h_cris 减 1。

反应模型的通用基本规则：
1） h_init 是反应的引发位点，也是 GPU 内线程计算的对象，h_cris 代表着是否是反应对象，对于自由基聚合反应和交换反应，h_cris=0 为可以反应的自由单体，h_cris=1 为不能反应的粒子。对于逐步聚合反应 h_cris 的值和 setMaxCris 的值决定反应位点和自由单体还有没有反应活性。
2） 反应位点 h_init=1 和反应位点之间不能够发生反应，
对于交换反应如果 A + B-C = A-B + C 和 A + B-C = A-C + B 这两种反应的可能性都存在，也就是说 reaction.setExchangePr ('A', 'B', 'C',0.1)，reaction.setExchangePr ('A','C','B',0.1)这种情况下如果同一个分子的 B 和 C 都是反应位点，程序会自动检测并跳出。